**LOADING AND PREPROCESSING**

**THE DATASET**

Introduction:

In this section of the code, we are loading and preparing the dataset for analysis. This involves importing necessary libraries, reading the dataset, exploring its contents, handling missing values, and visualizing relationships between variables.

Importing Libraries:

Code:

import pandas as pd

import numpy as np

import plotly.express as px

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.tree import DecisionTreeRegressor

Explanation:

Pandas (as pd): This library provides data structures and functions for efficiently manipulating large datasets.

numpy (as np): NumPy is used for numerical operations and provides support for arrays and matrices.

plotly.express (as px): Plotly Express is a high-level interface for creating various types of visualizations.

seaborn (as sns): Seaborn is a statistical data visualization library based on Matplotlib, providing a high-level interface for creating informative and attractive statistical graphics.

matplotlib.pyplot (as plt): Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations.

train\_test\_split: This function from scikit-learn helps split the dataset into training and testing sets for model evaluation.

DecisionTreeRegressor: This is a regression algorithm from scikit-learn used for modelling.

Loading the dataset:

Code:

data = pd.read\_csv("demand.csv")

Explanation:

pd.read\_csv(): This function from Pandas reads a CSV file into a DataFrame. In this case, it's reading a file named "demand.csv" and assigning it to the variable data.

At this point, we have successfully imported the necessary libraries and loaded the dataset into a Pandas DataFrame named data. The dataset is now ready for further exploration and analysis.

Displaying the First Few Rows:

Code:

print("HEAD:\n")

print(data.head())

Explanation:

**data.head():** This function displays the first few rows (default is 5) of the DataFrame, providing an initial look at the dataset's structure and content.

Output:

Checking for Missing Values:

Code:

print("\ncontains any null values or not:\n")

print(data.isnull().sum())

Explanation:

**data.isnull():** This returns a DataFrame of boolean values indicating whether each element in data is missing or not.

**sum():** This function sums up the True values (i.e., the missing values) for each column.This section prints out the number of missing values for each column in the dataset.

Output:

Handling Missing Values:

Code:

data = data.dropna()

Explanation:

**data.dropna():** This function removes rows with any missing values from the DataFrame and assigns the modified DataFrame back to data.

This step is taken to ensure that the dataset is clean and ready for analysis, as some machine learning algorithms do not handle missing values.

Total code for preprocessing:

import pandas as pd

import numpy as np

import plotly.express as px

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.tree import DecisionTreeRegressor

data = pd.read\_csv("demand.csv")

print("HEAD:\n")

print(data.head())

print("\ncontains any null values or not:\n")

print(data.isnull().sum())

data = data.dropna()

fig = px.scatter(data, x="Units Sold", y="Total Price",

size='Units Sold')

fig = px.scatter(data, x="Units Sold", y="Total Price", size='Units Sold', color\_discrete\_sequence=['red'])

fig = px.scatter(data, x="Units Sold", y="Total Price", size='Units Sold', template='plotly\_dark')

# Change the color of the scatter points

fig.update\_traces(marker=dict(color='green'))

fig.show()

print("correlation between the features of the dataset:\n")

print(data.corr())

correlations = data.corr(method='pearson')

plt.figure(figsize=(15, 12))

sns.heatmap(correlations, cmap="coolwarm", annot=True)

plt.show()

X = data[["Total Price", "Base Price"]]

y = data["Units Sold"]

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

model = DecisionTreeRegressor()

model.fit(X\_train, y\_train)

train\_score = model.score(X\_train, y\_train)

test\_score = model.score(X\_test, y\_test)

print("\nR-squared scores for Train and Test:\n")

print(f"Training R-squared score: {train\_score}")

print(f"Testing R-squared score: {test\_score}")

Output: